УДК 519.8 Вестник СПбГУ. Сер. 10. 2016. Вып. 4

Н. С. Григорьева

ЗАДАЧА МИНИМИЗАЦИИ МАКСИМАЛЬНОГО ВРЕМЕННОГО СМЕЩЕНИЯ ДЛЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОРОВ

Санкт-Петербургский государственный университет, Российская Федерация,
199034, Санкт-Петербург, Университетская наб., 7–9

Задача минимизации максимального временного смещения для составления расписаний для параллельных идентичных процессоров — классическая комбинаторная оптимизационная задача, имеет много приложений и является NP-трудной. Эта задача составления расписаний обозначается как $P|r_j|L_{\max}$ и формулируется следующим образом: задания должны быть выполнены на нескольких параллельных идентичных процессорах; требуется определить, где и когда каждое задание должно быть выполнено так, чтобы минимизировать максимальное смещение. Для каждого задания известны время поступления задания на выполнение, время выполнения и директивный срок, к которому задание должно быть закончено. Прерывания выполнения заданий не допускаются. Большинство разработок приближенных алгоритмов концентрируется на построении незадерживающих расписаний, в которых процессор не может простаивать, если есть готовое к выполнению задание. Но для построения оптимального расписания необходимо рассматривать такие допустимые расписания, в которых невынужденный простой процессора возможен. Цель данной статьи — представить приближенный алгоритм с невынужденными простоями ELS/IIT (в котором наибольшим приоритетом обладает задание с минимальным поздним началом) и метод ветвей и границ, который строит допустимое расписание с заданным значением максимального смещения. Для нахождения оптимального решения предлагается комбинировать метод ветвей и границ с бинарным поиском. Библиогр. 14 назв. Табл. 5.

Ключевые слова: параллельные процессоры, метод ветвей и границ, максимальное временное смещение.

N. S. Grigoreva

SCHEDULING PROBLEM TO MINIMIZE THE MAXIMUM LATENESS FOR PARALLEL PROCESSORS

St. Petersburg State University, 7–9, Universitetskaya nab.,
St. Petersburg, 199034, Russian Federation

The problem of minimizing the maximum lateness while scheduling tasks to parallel identical processors is a classical combinatorial optimization problem. It has many applications, and it is NP-hard. This problem relates to the scheduling problem and it is denoted by $P|r_j|L_{\max}$. The multiprocessor scheduling problem is defined as follows: tasks have to be executed on several parallel identical processors. We must find where and when each task will be executed, such that the maximum lateness is minimum. The duration, release time and due date of each task are known. Preemption on processors is not allowed. A lot of research in scheduling has concentrated on the construction of the nondelay schedule. A nondelay schedule is a feasible schedule in which no processor is kept idle at a time when it could begin processing a task. An inserted idle time schedule (IIT) as a feasible schedule in which the processor is kept idle at a time when it could begin processing a task. The goal of this paper is to propose an IIT schedule for the $P|r_j|L_{\max}$ problem. We propose an approximate IIT algorithm named ELS/IIT (earliest latest start/ inserted idle time) and depth first branch and bound algorithm, which produces

Григорьева Наталья Сергеевна — кандидат физико-математических наук, доцент; n.s.grig@gmail.com

Grigoreva Natalia Sergeevna — PhD of physical and mathematical sciences, associate professor; n.s.grig@gmail.com

DOI: 10.21638/11701/spbu10.2016.405

a feasible IIT (inserted idle time) schedule for a fixed maximum lateness L. The algorithm may be used in a binary search mode to find the smallest maximum lateness. The branch and bound algorithm is based on the earliest latest start/ inserted idle time branching strategy. Two new dominance criterions between decision nodes are used. A new method for evaluating unfeasible partial solutions was designed. To illustrate the effectiveness of this approach we tested algorithms on instances, which were randomly generated with the number of jobs from 50 to 300. Refs 14. Tables 5.

Keywords: parallel identical processors, branch and bound algorithm, maximum lateness.

1. Введение. Задачи составления расписаний для параллельных процессоров — один из важных разделов теории расписаний [1]. Максимальное временное смещение $L_{\max}$, которое будет описано ниже, — широко используемый критерий качества расписания. Следуя схеме, предложенной Р. Грехемом и др. [2], задача минимизации максимального временного смещения $L_{\max}$ для множества заданий, которые выполняются на параллельных идентичных процессорах, обозначается как $P|r_j|L_{\max}$, имеет много приложений и является NP-трудной в сильном смысле [3]. Ряд работ посвящен различным вариантам данной задачи: чаще всего рассматривались модели с единичными длительностями заданий $P|r_j, p_j = 1|L_{\max}$ [4], в которых все задания выполнялись на одном процессоре $1|r_j|L_{\max}$, и задачи с разрешением прерываний при выполнении заданий [5].

В большинстве исследований описываются незадерживающие расписания, определенные К. Бейкером [6] как допустимые расписания, где процессор не может простаивать, если есть задания, которые процессор может выполнять, и класс списочных алгоритмов, строящих незадерживающие расписания. Но известно, что оптимальные расписания могут содержать невынужденные простои, и в этих случаях списочные алгоритмы не могут построить оптимальное расписание. Расписания с невынужденными простоями (inserted idle time — IIT) были определены в [7] как допустимые, в которых процессор может простаивать при наличии готовых к выполнению заданий. В [7] дан обзор литературы для задач составления расписаний, в которых актуально использование невынужденных простоев, в нем большинство рассмотренных задач относится к моделям с одним процессором. Для многих моделей разрешение невынужденных простоев не приводит к улучшению расписаний, этот подход не продуктивен для задач с одинаковой длительностью заданий и для расписаний с прерываниями. Для задачи $1|r_j|L_{\max}$ в [8] был представлен алгоритм, позволяющий вставлять невынужденные простои в расписание.

В работе [9] разработан алгоритм построения расписаний с невынужденными простоями для задачи $P|prec|C_{\max}$, в которой на множестве заданий задано отношение частичного порядка и целевой функциией служит длина расписания. Для этой задачи предлагались приближенный жадный алгоритм и метод ветвей и границ, конструирующий допустимое IIT расписание с заданной длиной. Для получения оптимального решения задачи метод ветвей и границ комбинировался с бинарным поиском. Тестирование таких алгоритмов показало продуктивность данных подходов.

В [10] для задачи $P|r_j|L_{\max}$ были предложены приближенный алгоритм составления расписания и общая схема метода ветвей и границ.

Цель данной статьи — применить основные идеи методов из [9] для задачи $P|r_j|L_{\max}$ и установить для нее эффективность алгоримов с невынужденными простоями, для чего исследуются несколько вариантов приближенных алгоритмов минимизации максимального временного смещения и приводится детальное описание метода ветвей и границ, который конструирует допустимое IIT расписание

с заданным временным смещением L. Для получения оптимального решения задачи метод ветвей и границ комбинируется с бинарным поиском.

Для проверки эффективности алгоритмов с невынужденными простоями для задачи $P|r_j|L_{\max}$ мы протестировали их на случайно сгенерированных тестовых примерах.

2. Постановка задачи. Имеются множество заданий $U = \{u_1, u_2, \ldots, u_n\}$ и m параллельных идентичных процессоров для их выполнения. Для каждого задания $u_i \in U$ заданы время выполнения $t(u_i)$, время поступления в систему $r(u_i)$ и директивный срок его завершения $D(u_i)$. Любое задание может выполняться на любом процессоре, и каждый процессор может выполнять не более одного задания в каждый момент времени. Прерывания выполнения заданий не допускаются.

Составить расписание для множества заданий U — значит, назначить каждому заданию $u_i \in U$ время начала выполнения $\tau(u_i)$ и процессор $num(u_i)$. В качестве характеристики расписания будем рассматривать максимальное временное смещение, которое для расписания S определяется по формуле

$$L_{\max} = \max\{\tau(u_i) + t(u_i) - D(u_i) | u_i \in U\}.$$

Требуется составить расписание, минимизирующее $L_{\max}$.

Сначала опишем и сравним приближенные жадные алгоритмы, затем, комбинируя их и метод ветвей и границ, представим ВВ/ИТ алгоритм, который позволяет получить оптимальные решения для многопроцессорной задачи составления расписаний с директивными сроками.

3. Приближенные жадные алгоритмы. Рассмотрим пять вариантов жадных приближенных алгоритмов. На каждом шаге жадного алгоритма одно из заданий будет окончательно устанавливаться на процессор. Для задания u_i известны раннее возможное время начала задания $r(u_i)$ и позднее время начала задания $v_{\max}(u_i) = D(u_i) - t(u_i)$, при котором задание будет закончено до наступления директивного срока. Если k заданий уже поставлено в расписание, то обозначим множество таких заданий через S_k.

Определим две очевидные нижние границы целевой функции:

$$LB1 = \max\{r(u_i) + t(u_i) - D(u_i) | u_i \in U\},$$
$$LB2 = \max\{\lceil \sum_{i=1}^{n} t(u_i)/m \rceil - D_{\max}\}.$$

Тогда нижняя граница $LB = \max\{LB1, LB2\}$. Будем считать, что для пустого решения S_0 нижняя оценка целевой функции $L(S_0)$ равна LB. В процессе построения расписания будем корректировать нижнюю оценку целевой функции следующим образом: при установке задания u_0 на процессор в момент времени $\tau(u_0)$ нижняя оценка пересчитывается по формуле

$$LB(S_{k+1}) = \max\{LB(S_k), \tau(u_0) + t(u_0) - D(u_0)\}.$$

Для построения приближенного алгоритма надо задать стратегию выбора очередного задания для включения в расписание и разрешить или запретить невынужденные простои. Одним из классических вариантов жадного алгоритма является выбор очередного задания с минимальным директивным сроком — EDD (eailest due

date) алгоритм [11]. Для частного случая, при котором все задания имеют одинаковые времена поступления, этот алгоритм строит оптимальное расписание [11]. Для задачи с частичным порядком в [9] было предложено и хорошо себя зарекомендовало правило выбора задания с минимальным поздним началом. Будем называть такое задание критическим, а алгоритм, реализующий данную стратегию, — ELS (eailest latest start) алгоритмом.

При любом выборе приоритета заданий можно строить незадерживающие расписания (nodelay — ND) или расписания с простоями (IIT). Комбинируя разные подходы, получим четыре варианта жадного алгоритма. Если перед началом выполнения критического задания возникает невынужденный простой, то, для того чтобы исключить нерациональное использование времени процессора, будем подбирать другое задание, которое можно выполнять во время простоя процессора. Таким образом, выбор задания будет осуществляться в два этапа. Возможны несколько различных стратегий заполнения времени простоя, в [9] для задачи с частичным порядком на множестве заданий предлагалось не увеличивать время начала критического задания. Для задачи $P|r_j|L_{\max}$ мы модифицировали алгоритм, разрешив сдвигать время начала критического задания так, чтобы не увеличивать нижнюю оценку целевой функции. Опишем два варианта алгоритма.

Пусть k заданий уже поставлено в расписание, нам известны $time_k[1:m]$ времена освобождения процессоров после выполнения всех заданий из множества S_k и текущая нижняя оценка целевой функции $L(S_k)$. Приближенное решение конструируется алгоритмами ELSM/IIT и ELS/IIT следующим образом.

Алгоритмы ELS/IIT и ELSM/IIT:

1. Найдем процессор l_0 такой, что $time_k[l_0] = \min\{time_k[i] | i \in 1:m\}$, обозначим $t_{\min}(k) = time_k[l_0]$.

2. Выберем задание u_0 такое, что $v_{\max}(u_0) = \min\{v_{\max}(u_i) | u_i \notin S_k\}$.

3. Если $idle(u_0) = r(u_0) - t_{\min}(k) > 0$, то выберем задание $u^* \notin S_k$, которое может выполняться в период простоя процессора l_0 без значительного увеличения времени начала выполнения задания u_0, а именно (вариант ELS)

$$v_{\max}(u^*) = \min\{v_{\max}(u_i) | r(u_i) + t(u_i) \leqslant r(u_0), u_i \notin S_k\}$$

или (вариант ELSM)

$$v_{\max}(u^*) = \min\{v_{\max}(u_i) | r(u_i) + t(u_i) \leqslant v_{\max}(u_0) + LB(S_k), u_i \notin S_k\}.$$

4. Если задание u^* найдено, то примем, что $u(k) = u^*$, иначе положим $u(k) := u_0$.

5. Назначим на процессор l_0 задание $u(k)$.

6. Пересчитаем нижнюю оценку целевой функции (вариант ELSM)

$$LB(S_{k+1}) = \max\{LB(S_k), \tau(u(k)) + t(u(k)) - D(u(k))\}.$$

В вычислительном эксперименте сравним расписания, построенные алгоритмами ELS/IIT и EDD/IIT с расписаниями, построенными алгоритмами без невынужденных простоев ELS/ND и EDD/ND. Приближенное расписание строится алгоритмом ELS/ND следующим образом.

Алгоритм ELS/ND:

1. Определить процессор l_0 такой, что $time_k[l_0] = \min\{time_k[i] | i \in 1 : m\}$, обозначим $t_{\min}(k) = time_k[l_0]$.

2. Выбрать задание u_0 такое, что $v_{\max}(u_0) = \min\{v_{\max}(u_i) | u_i \notin S_k\}$.

3. Если $idle(u_0) = r(u_0) - t_{\min}(k) > 0$, то выбрать задание $u^* \notin S_k$, которое может выполняться без простоя:

$$v_{\max}(u^*) = \min\{v_{\max}(u_i) | r(u_i) \leqslant t_{\min}(k), u_i \notin S_k\}.$$

4. Если задание u^* найдено, то назначить на процессор l_0 задание u^*, иначе — задание u_0.

Алгоритмы различаются в шаге 3, ELS/ND алгоритм является списочным, в котором процессор не может простаивать, если есть готовое к выполнению задание. Вычислительная сложность предложенных алгоритмов $O(n^2)$.

Для иллюстрации различия алгоритмов рассмотрим пример [11], в котором алгоритм EDD/ND строит самое плохое расписание. Пусть имеются m^2+m+1 задание, которые выполняются на m параллельных процессорах. Среди них есть m заданий длины $t_{\max}$, для которых времена поступления равны нулю, а директивные сроки — большому числу K. Также имеются m^2 заданий длины $t_{\max}/m$ и одно задание длины $t_{\max}$, для которых времена поступления равны r, а директивные сроки — нулю.

Для такой системы заданий алгоритм EDD/ND построит расписание с максимальным временным смещением $L_{\max} = 3t_{\max}$, алгоритм EDD/IIT — расписание с $L_{\max} = 3t_{\max} + r$, а алгоритм ELS/ND — с $L_{\max} = 2t_{\max} + t_{\max}/m$. Алгоритмы ELS/IIT и ELSM/IIT построят оптимальное расписание со значением целевой функции $L_{\max} = t_{\max} + t_{\max}/m + r$. Следует заметить, что если для всех заданий $t(u_i) = 1$, то все алгоритмы построят одно и тоже оптимальное расписание. В случае, когда все времена поступлений и все директивные сроки одинаковы, простоев в расписаниях не будет. При этом алгоритм EDD будет ставить задания в расписание в произвольном порядке, а алгоритм ELS будет работать как алгоритм LPT, в нем сначала ставятся задания с наибольшим временем выполнения, и он имеет гарантированную оценку точности, равную $4/3 - 1/3m$ [2]. Сравнение работы пяти жадных алгоритмов приведены в п. 6. По их результатам для метода ветвей и границ была выбрана стратегия выбора задания ELS/IIT.

4. Алгоритм построения оптимального расписания. Для построения оптимального расписания предлагается комбинировать бинарный поиск и метод ветвей и границ. Такой подход для ряда минимаксных задач был предложен И. В. Романовским и Н. П. Христовой [12], в частности рассматривалась задача составления расписаний для параллельных процессоров $P||C_{\max}$.

Обозначим L_{opt} максимальное временное смещение оптимального расписания. Необходимо определить интервал $(a, b]$ такой, что $a < L_{\text{opt}} \leqslant b$.

Нижняя граница значения целевой функции LB была установлена выше, положим $a := LB - 1$. Далее опишем процедуру построения оценки частичного решения, которая позволяет уточнять нижнюю оценку целевой функции. Верхняя оценка целевой функции $b = \lceil \sum_{i=1}^{n} t(u_i)/m \rceil + r_{\max} + t_{\max} - D_{\min}$, где $r_{\max} = \max\{r(u_i) | u_i \in U\}$, $t_{\max} = \max\{t(u_i) | u_i \in U\}$ и $D_{\min} = \min\{D(u_i) | u_i \in U\}$ [13].

Для уточнения верхней границы рекомендуется применить приближенный метод решения задачи ELSM/IIT и выбрать величину целевой функции, полученную этим методом, в качестве b. Тогда $L_{\text{opt}} \in (a, b]$.

Выберем $z = \lceil (a+b)/2 \rceil$ и используем метод ветвей и границ для построения допустимого расписания $BB(U, D+z, m; S)$ со значением максимального временного смещения, не превосходящим z. Если будет построено такое расписание, то выберем интервал $(a, z]$, в противном случае выберем интервал $(z, b]$ и повторим процесс.

Алгоритм OPT SCHEDULE $(U; S_{\text{opt}}, L_{\text{opt}})$:

1. Вычислить a и b.
2. Пока $b - a > 1$ выполнять
3. Начало цикла.
4. Положить $z := \lceil (a+b)/2 \rceil$. Пересчитать директивные сроки $D(u_i)$, положив $D^*(u_i) = D(u_i) + z$, максимальный директивный срок $D^*_{\max} = \max\{D^*(u_i) | u_i \in U\}$ и поздние времена начала заданий $v_{\max}(u_i) = D^*(u_i) - t(u_i)$.
5. Использовать процедуру метода ветвей и границ $BB(U, D^*; S, L_S)$ для построения допустимого расписания.
6. Если построено допустимое решение S, то найти величину целевой функции $L_S = \max\{\tau(u_i) + t(u_i) - D(u_i) | u_i \in U\}$, запомнить расписание $S_{\text{rec}} := S$ и значение целевой функции $L_{\text{rec}} := L_S$, положить $b := L_S$. Если допустимого решения не существует, то положить $a := z$.
7. Конец цикла.
8. $S_{\text{opt}} := S_{\text{rec}}$ и $L_{\text{opt}} := L_{\text{rec}}$.

5. Метод ветвей и границ построения допустимого расписания $BB(U, D^*; S, L_S)$**.** Этот метод строит допустимое IIT расписание, в котором каждое задание $u_i \in U$ должно заканчиваться не позже модифицированного директивного срока $D^*(u_i)$. В нем применяется односторонний обход дерева перебора. Для формального описания метода ветвей и границ необходимо определить частичное решение, применить метод продолжения частичного решения (шаг вперед), метод выяснения недопустимости частичного решения и шаг назад, который восстанавливает предыдущее частичное решение.

Будем представлять частичное решение как частичную перестановку заданий. Для каждой перестановки заданий $\pi = (u_{i_1}, u_{i_2}, \ldots, u_{i_n})$ можно построить расписание S_π следующим образом: на каждом шаге находится процессор, который раньше всех освобождается. Затем очередное задание устанавливается на этот процессор, и его выполнение начинается в минимально возможный момент времени. Таким образом, каждая перестановка будет однозначно определять расписание S_π. Частичное решение σ_k, где k — число заданий, будем представлять как частичную перестановку $\sigma_k = (u_{i_1}, u_{i_2}, \ldots, u_{i_k})$, которая задает частичное расписание.

Определение 1. *Расписание $\gamma_n = (l_1, l_2, \ldots, l_n)$ называется продолжением частичного расписания $\sigma_k = (q_1, q_2, \ldots, q_k)$, если $l_1 = q_1,\ l_2 = q_2,\ \ldots,\ l_k = q_k$.*

Определение 2. *Частичное решение σ_k называется допустимым, если существует его продолжение σ_k, которое является допустимым расписанием.*

Для каждого задания u_i известны ранний возможный срок начала задания $r(u_i)$ и поздний срок начала задания $v_{\max}(u_i) = D^*(u_i) - t(u_i)$. Для построения допустимого расписания необходимо, чтобы для каждого задания $u_i \in U$ время начала его выполнения $\tau(u_i)$ удовлетворяло неравенству

$$r(u_i) \leqslant \tau(u_i) \leqslant v_{\max}(u_i).$$

Модифицированные директивные сроки задают для допустимого расписания S суммарное возможное время простоев I всех процессоров. Для m процессоров суммарное время простоев вычисляется по формуле

$$I = m \cdot D^*_{\max} - \sum_{i=1}^{n} t(u_i).$$

На каждом уровне перебора k будем формировать множество заданий U_k, которые будем называть готовыми заданиями. Это те задания, которые следует добавить к частичному решению σ_{k-1} так, чтобы проверить все возможные допустимые продолжения частичного решения. Так как метод ветвей и границ должен изучать все перестановки заданий, то любое задание, еще не включенное в частичное решение на данном уровне перебора, необходимо проверить. Но очевидно, если простой перед выполнением выбранного задания будет слишком велик, то построить допустимое расписание, в котором все задания должны быть выполнены до момента времени $D^*_{\max}$, не удастся. Следующее определение готового задания исключает из рассмотрения те задания, для которых простой превышает возможное значение. Уменьшение множества готовых заданий дает возможность сократить перебор.

Определение 3. *Задание $u \notin \sigma_k$ называется готовым заданием на уровне k, если время его поступления $r(u)$ удовлетворяет неравенству*

$$r(u) - t_{\min}(k) \leqslant I - \sum_{u \in \sigma_k} idle(u_i).$$

Выбор задания из множества готовых заданий выполняется так же, как в приближенном алгоритме ELS/IIT. Алгоритм останавливается, как только удается построить полное допустимое решение. Если обнаружено, что частичное решение недопустимо, последнее назначенное задание удаляется из частичного решения, исключается из множества U_k и процедурой выбора задания выбирается новое задание для установки на процессор. Если множество U_k пусто, то отменяется последнее назначенное задание, и алгоритм поднимается еще на один уровень вверх в дереве перебора.

Основной путь сокращения перебора в методе ветвей и границ — это раннее определение недопустимости частичного решения и исключение неперспективных заданий из множества готовых.

Определение 4. *Пусть задание $u_{\text{cr}} \notin \sigma_k$ удовлетворяет условию $v_{\max}(u_{\text{cr}}) = \min\{v_{\max}(u) | u \notin \sigma_k\}$. Задание $u_{\text{cr}} \notin \sigma_k$ будем называть опоздавшим для частичного решения σ_k, если $v_{\max}(u_{\text{cr}}) < t_{\min}(k)$.*

Сформулируем и докажем несколько правил определения недопустимых частичных решений.

Лемма 1. *Если опоздавшее задание u_{cr} для частичного решения σ_k существует, то частичное решение σ_k недопустимо.*

Доказательство очевидно. □

Сформулируем условие, которое позволяет после обнаружения опоздавшего задания удалить дополнительно некоторые недопустимые частичные решения и сократить число заданий в множестве U_k.

Лемма 2. *Если для частичного решения* $\sigma_k = \sigma_{k-1} \cup u_k$ *существует опоздавшее задание* u_{cr}*, тогда для любого задания* u *такого, что* $\max\{t_{\min}(k-1), r(u)\} + t(u) > v_{\max}(u_{\text{cr}})$*, частичное решение* $\sigma_{k-1} \cup u$ *недопустимо.*

Доказательство. Пусть $t_{\min}(k)$ есть самый ранний момент времени окончания выполнения всех заданий из частичного решения σ_k. Пусть l_0 — самый ранний освободившийся процессор, а именно процессор, для которого выполнено $time_k[l_0] = t_{\min}(k)$. Если задание u_{cr} является опоздавшим, то верно $v_{\max}(u_{\text{cr}}) < t_{\min}(k)$.

Следовательно, нельзя назначить задание u_{cr} на любой другой процессор, кроме l_0. После удаления последнего поставленного в расписание задания u_k алгоритм возвращается к частичному решению σ_{k-1}. Пусть процессор l_1 — самый ранний освободившийся процессор для частичного решения σ_{k-1}, и он заканчивает работу в момент времени $t_{\min}(k-1)$. (Возможно, что $l_0 = l_1$.) Если добавить задание u к частичному решению σ_{k-1} на шаге $k-1$, то на следующем шаге k следует назначить задание u_{cr} только на тот же самый процессор l_1. Таким образом, должно быть выполнено

$$\max\{t_{\min}(k-1), r(u)\} + t(u) \leqslant v_{\max}(u_{\text{cr}}).$$

В противном случае частичное решение $\sigma_{k-1} \cup u$ будет недопустимым, так как для него задание u_{cr} будет опоздавшим. □

Рассмотрим ситуацию, когда два еще не назначенных задания блокируют друг друга, и, следовательно, такое частичное решение не может иметь допустимого продолжения.

Лемма 3. *Если существует опоздавшее задание* u_{cr} *для частичного решения* $\sigma_k = \sigma_{k-1} \cup u_k$ *и выполнено* $\max\{t_{\min}(k-1), r(u_{\text{cr}})\} + t(u_{\text{cr}}) > v_{\max}(u_k)$*, то частичное решение* σ_{k-1} *недопустимо.*

Доказательство. Рассмотрим два частичных решения σ_k и σ_{k-1}. Пусть процессор l_0 — самый ранний освободившийся процессор для частичного решения σ_k на шаге k и l_1 — самый ранний освободившийся процессор для частичного решения σ_{k-1}.

Рассмотрим следующие случаи.

1. Пусть $v_{\max}(u_k) \leqslant v_{\max}(u_{\text{cr}})$. Если u_{cr} — опоздавшее задание, то $v_{\max}(u_{\text{cr}}) < t_{\min}(k)$. Таким образом, $v_{\max}(u_k) < t_{\min}(k)$, и назначить задание u_k на процессор l_0 тоже нельзя.

После удаления задания u_k задание u_{cr} назначается на процессор l_1, который заканчивает выполнение заданий в момент времени $time_k[l_1] = \max\{t_{\min}(k-1), r(u_{\text{cr}})\} + t(u_{\text{cr}})$. По условию леммы $\max\{t_{\min}(k-1), r(u_{\text{cr}})\} + t(u_{\text{cr}}) > v_{\max}(u_k)$. Следовательно, задание u_k невозможно назначить на процессор l_1 и u_k будет опоздавшим заданием для частичного решения $\sigma_k = \sigma_{k-1} \cup u_{\text{cr}}$.

2. Если $v_{\max}(u_k) > v_{\max}(u_{\text{cr}})$, то возможны два варианта. Первый, когда частичное решение $\sigma_{k-1} \cup u_{\text{cr}}$ было проверено раньше и оказалось недопустимым. Для любого решения $\sigma_{k-1} \cup u$, где $u \in U_k$, задания u_k и u_{cr} будут опоздавшими. Во втором задание u_k было поставлено в расписание раньше u_{cr}, если заполняло простой перед началом выполнения задания u_{cr}. Но тогда u_{cr} не может быть опоздавшим заданием по свойствам алгоритма. Частичное решение $\sigma_k = \sigma_{k-1} \cup u$ недопустимо для всех $u \in U_k$, таким образом, частичное решение σ_{k-1} недопустимо. □

Следующий способ определения недопустимого частичного решения основан на сравнении суммарной потребности заданий в ресурсах и общей мощности процессоров на некотором интервале. В этом случае модифицируем алгоритм для определения

интервала концентрации [14] для полного решения, применим его к частичному решению σ_k. Времена поступления заданий $r(u)$ отсортируем по неубыванию и выделим из них различные значения, которые сохраним в массиве $rd(i)$, где $i \in 1 : n1$. Аналогично отсортируем директивные сроки $D^*(u)$ и разные значения сохраним в массиве $Dd(i)$, где $i \in 1 : n2$. При $n1 = n2 = 1$ такая задача будет эквивалентна задаче $P||C_{\max}$.

Выберем временной интервал $[t_1, t_2] \subseteq [t_{\min}(k), D^*_{\max}]$, где $t_1 < t_2$. Моментами времени t_1 будем считать элементы массива $rd(i)$, а значение t_2 будем выбирать из массива Dd. Обозначим через $MP(t_1, t_2)$ общее время свободных процессоров во временном интервале $[t_1, t_2]$. Тогда

$$MP(t_1, t_2) = \sum_{i=1}^{m} \max\{0, (t_2 - \max\{t_1, time_k[i]\})\}.$$

Для всех заданий, не входящих в частичное решение $u_i \notin \sigma_k$, определим минимальное время начала выполнения: $v_k(u_i) = \max\{r(u_i), t_{\min}(k)\}$, а затем найдем минимальный интервал его выполнения, пересекающийся с интервалом $[t_1, t_2]$. Для этого вычислим

$$x_k(u_i) = [v_k(u_i), v_k(u_i) + t(u_i)] \cap [t_1, t_2],$$
$$y(u_i) = [v_{\max}(u_i), v_{\max}(u_i) + t(u_i)] \cap [t_1, t_2].$$

Обозначим $L(x_k(u_i))$ и $L(y(u_i))$ дли́ны временных интервалов $x_k(u_i)$ и $y(u_i)$ соответственно. Пусть $M_k(t_1, t_2)$ — общее минимальное время, которое требуется всем, еще не включенным в частичное расписание, заданиям в интервале $[t_1, t_2]$, тогда

$$M_k(t_1, t_2) = \sum_{u_i \notin \sigma_k} \min\{L(x_k(u_i)), L(y(u_i))\}.$$

Выделив наиболее загруженные заданиями временные интервалы, можно соотнести суммарную потребность в ресурсах на этих интервалах с общим свободным временем имеющихся процессоров.

Пусть

$$est(\sigma_k) = \max_{[t_1,t_2] \in [t_{\min}(k), D^*_{\max}]} \{M_k(t_1, t_2) - MP(t_1, t_2)\}.$$

Эту процедуру будем применять также для уточнения нижней оценки целевой функции, которая вычисляется до начала метода ветвей и границ. Пусть нижняя оценка целевой функции LB определена, как указано выше. Пересчитаем исходные директивные сроки $D(u)$, положив $D^*(u_i) = D(u_i) + LB$. Для пустого частичного решения σ_0 положим $t_{\min} = 0$ и найдем $est(\sigma_0)$. При $est(\sigma_0) > 0$ нижняя оценка целевой функции LB должна быть увеличена.

Лемма 4. *Если $est(\sigma_0) > 0$, то нижняя оценка целевой функции $LBM = LB + \lceil est(\sigma_0)/m \rceil$. Если $est(\sigma_k) > 0$, то частичное решение σ_k недопустимо.*

Представим псевдокод метода ветвей и границ построения допустимого расписания $BB(U, D^*; S, L_S)$:

Algorithm 1 : ВВ/ИТ алгоритм

Положить $k := 1; time[i] = 0; i \in 1 : m$; $\sigma_0 = \emptyset$.
while $(k > 0)$ $(k < n + 1)$ **do**
  Определить процессор l_0 такой, что $t_{\min}(l_0) = \min\{(time_k[i] | i \in 1, ..., m)\}$.
  Найти задание u_{cr} такое, что $v_{\max}(u_{\text{cr}}) = \min\{v_{\max}(u) | u \notin \sigma_{k-1}\}$.
  if $v_{\max}(u_{\text{cr}}) \geqslant t_{\min}(l_0)$ **then**
    Вычислить $EST = est(\sigma_{k-1})$.
    if $EST \leqslant 0$ **then**
      Выбрать задание u_0 аналогично алгоритму ELS/ИТ $Select(U_k, t_{\min}(k); u_0)$.
      Назначить задание u_0 на процессор l_0 и создать частичное решение $\sigma_k = \sigma_{k-1} \cup u_0$.
    else
      Выполнить шаг назад и восстановить частичное решение σ_{k-1}.
    end if
  else
    Определить опоздавшее задание u_{cr}. Удалить все недопустимые решения по леммам 2 и 3.
  end if
end while
if $k = 0$, **then**
  Максимальное временное смещение оптимального расписания больше, чем L_S.
end if
if $k = n$, **then**
  Найдено допустимое решение $S = \sigma_n$, и его максимальное временное смещение меньше либо равно L_S.
end if

6. Вычислительные результаты. Представим численные результаты тестирования предложенных приближенных алгоритмов и метода ветвей и границ ВВ/ИТ.

Для тестирования алгоритмов были использованы тесты из Standard Task Graph Set, доступные на сайте http://www.kasahara.elec.waseda.ac.jp/schedule/. Standard Task Graph Set является своего рода эталоном для оценки алгоритмов составления многопроцессорных расписаний и включает наборы как случайно сгенерированных тестов, так и тестов, смоделированных из реальных прикладных задач. В этом наборе тестов заданы отношения частичного порядка на множестве заданий U, которые случайно генерируются с помощью четырех различных алгоритмов, и времена выполнения заданий, генерирующиеся также с использованием нескольких методов.

В задаче минимизации максимального временного смещения задания множества U независимы, но для каждого задания должны быть известны время поступления и директивный срок. Поэтому тесты адаптировались для рассматриваемой задачи следующим образом. По графу, задающему отношения частичного порядка на множестве заданий U, для каждого задания вычислялся самый ранний возможный срок начала задания, и он брался в качестве времени поступления задания $r(u)$. Для каждого задания вычислялось наиболее позднее время его окончания и принималось в качестве директивного срока $D(u)$. Каждая серия тестов в данном наборе состоит из 180 примеров. При этом всюду строились оптимальные расписания для 2, 4 и 8 процессоров. Были рассмотрены тесты из Standard Task Graph Set для $n = 50$, $n = 100$ и $n = 300$, где n — количество заданий.

Количество итераций для поиска допустимых решений методом ветвей и границ было ограничено. Если допустимое расписание S с заданным максимальным временным смещением L для m процессоров не удавалось построить за 20 000 итераций, то предполагалось, что такое расписание не существует, и целевая функция L увеличивалась. Такой подход позволил построить расписания для всех тестовых задач, но оставался открытым вопрос: построено оптимальное решение или нет?

Результаты вычислительных экспериментов приведены в табл. 1–5. Первая колонка во всех таблицах содержит число заданий n, вторая — количество процессоров m.

Прежде всего была проверена эффективность пяти вариантов приближенных алгоритмов. В табл. 1 приведено среднее относительное отклонение от нижней оценки целевой функции $RT = (L_{\max} - LB)/LB$ для пяти рассмотренных алгоритмов.

Таблица 1. **Средние данные для приближенных алгоритмов**

n	m	ELSM/IIT	ELS/IIT	ELS/ND	EDD/IIT	EDD/ND
100	2	0.0001	0.0001	0.001	0.002	0.002
100	4	0.639	0.710	0.816	0.897	0.775
100	8	0.369	0.364	0.497	1.111	0.912
300	2	0.001	0.001	0.002	0.005	0.004
300	4	0.799	0.801	0.967	2.624	1.289
300	8	0.442	0.489	0.501	6.982	1.326
Среднее		0.374	0.394	0.463	1.995	0.717

Из табл. 1 следует, что лучшие приближенные расписания получены модифицированным алгоритмом ELSM. Правило выбора задания EDD существенно проигрывает правилу выбора задания с минимальным поздним началом, поэтому в методе ветвей и границ правило EDD не используется. В методе ветвей и границ, строящем допустимое расписание, нельзя увеличивать директивные сроки, потому выбор задания будет осуществляться стратегией ELS, для которой проверяются варианты построения расписания с простоями и без простоев.

Таблица 2 показывает результаты работы приближенного алгоритма ELSM/IIT.

Таблица 2. **Относительная погрешность для алгоритма ELSM/IIT**

n	m	N_{opt}	$RT < 0.05$	$RT < 0.1$	$RT > 0.1$
100	2	79.1	17.3	2.9	0.7
100	4	62.4	22.4	3.4	13.1
100	8	76.2	12.1	2.1	9.6
300	2	72.1	27.9	0	0
300	4	43.3	37.2	4.4	15.1
300	8	65.1	18.3	3.1	11.5
Среднее		66.3	22.53	2.84	8.33

Колонка N_{opt} содержит количество тестов (в процентах), в которых были получены оптимальные решения. Следующая колонка — это число тестов (в процентах), в которых были получены приближенные решения со средним относительным отклонением не больше, чем 0.05, но для которых не была доказана оптимальность решений из-за ограничения на число итераций. Но эти решения могут быть оптимальными. Две следующие колонки содержат число тестов, в которых $RT \in (0.05, 0.1]$ и $RT > 0.1$ соответственно. Приближенные решения с относительной погрешностью RT менее чем 10% были получены приближенным ELSM/IIT алгоритмом в 91.5% тестов. Оптимальные решения были получены в среднем в 66.3% тестов.

В табл. 3 приведены результаты сравнения эффективности разных способов определения недопустимых частичных решений. В ней EST — число частичных решений, удаленных по лемме 4, $LATE$ — число частичных решений, удаленных при обнаружении опоздавшего задания, DEL — число частичных решений, удаленных по леммам 2 и 3, $ITER$ — число итераций.

Таблица 3. **Средние данные для удаления недопустимых решений**

n	m	$EST/ITER$	$LATE/ITER$	$DEL/ITER$
100	2	0.0610	0.081	0.054
100	4	0.1590	0.063	0.028
100	8	0.2561	0.043	0.011
300	2	0.0120	0.080	0.063
300	4	0.1480	0.079	0.021
300	8	0.2431	0.085	0.009
Среднее		0.147	0.071	0.039

Как вытекает из табл. 3, условия леммы 4 позволяют удалить 14% недопустимых частичных решений, условия лемм 2 и 3–4 и 7% недопустимых частичных решений удаляются при обнаружении опоздавшего задания. Вычислительный эксперимент подтвердил, что применение всех предложенных правил удаления недопустимых частичных решений приводит к сокращению перебора в методе ветвей и границ.

В табл. 4 приведены результаты сравнения расписаний, полученных двумя вариантами метода ветвей и границ, которые отличаются процедурой выбора задания для продолжения частичного решения. Выбор очередного задания для продолжения частичного решения в методе ветвей и границ BB/IIT выполняется процедурой ELS/IIT, а в методе BB/ND — процедурой ELS/ND. В первом варианте сначала строятся расписания с простоями, а во втором более приоритетны незадерживающие расписания.

Также было проведено сравнение максимального временного смещения $L(\mathrm{IIT})$ для расписания, полученного методом ветвей и границ BB/IIT, и максимального временного смещения $L(\mathrm{ND})$ для расписания, построенного в результате работы BB/ND алгоритма (см. табл. 4).

Таблица 4. **Средние данные для BB/IIT и BB/ND алгоритмов**

n	m	BIIT	BND	NEQ	$L(\mathrm{ND})/L(\mathrm{IIT})$
100	2	3.1	0	96.8	1.02
100	4	51.1	0	48.9	1.04
100	8	59.2	0	41.8	1.14
300	2	11.8	1.8	86.2	1.02
300	4	18.6	21.8	60.5	1.05
300	8	19.1	18.4	61.8	1.03
Среднее		27.13	7	65.87	1.05

Третья колонка табл. 4 содержит число лучших решений (в процентах) для алгоритма BB/IIT, четвертая — число лучших решений для алгоритма BB/ND, пятая — число одинаковых решений для двух алгоритмов. Для всех серий тестов вычислялось отношение целевой функции $L(\mathrm{ND})$, полученной алгоритмом BB/ND, к значению целевой функции $L(\mathrm{IIT})$, полученной алгоритмом BB/IIT. Последняя колонка содержит среднее значение отношения $L(\mathrm{ND})/L(\mathrm{IIT})$.

Рассматривались только те случаи, в которых целевая функция не равна нулю. Из табл. 4 следует, что в 65% тестов алгоритмы дают одинаковое значение целевой функции, но лучших решений существенно больше в среднем у метода с простоями (27% против 7%). Для группы тестов из 300 заданий для 4 и 8 процессоров количество лучших решений примерно одинаково для обоих рассматриваемых алгоритмов.

Среднее относительное отклонение от нижней оценки целевой функции RT для расписаний, построенных методом ветвей и границ ВВ/IIT, представлено в табл. 5. Из нее вытекает, что оптимальные решения методом ветвей и границ были получены в 77.83% тестов (в среднем), а для 87% тестов приближенные решения имели относительную погрешность менее 5%.

Таблица 5. **Относительная погрешность для метода ветвей и границ ВВ/IIT**

n	m	N_{opt}	$RT < 0.05$	$RT < 0.1$	$RT > 0.1$
100	2	96	1	3	1
100	4	73	11	5	10
100	8	76	12	4	8
300	2	92	7	1	0
300	4	66	12.5	11	11
300	8	64	13	2	21
Среднее		77.83	9.25	4.33	8.5

7. Заключение. В этой работе рассматривались задача составления расписания $P|r_j|L_{\max}$ и алгоритмы ее решения, в которых разрешается допускать в расписании невынужденные простои процессоров. Для такой задачи были предложены новый жадный приближенный IIT алгоритм и алгоритм, в котором метод ветвей и границ для построения допустимого расписания комбинируется с бинарным поиском. Приближенные решения с относительной погрешностью $RT < 10\%$ были получены приближенным ESTM/IIT алгоритмом в 90% тестов, оптимальные решения — в среднем в 66% тестов.

Проведенный вычислительный эксперимент показал, что построенные методом ветвей и границ расписания с невынужденными простоями успешно конкурируют с классическими незадерживающими расписаниями. Примерно в 20% тестов расписания с простоями имели лучшее значение целевой функции. Но с увеличением количества заданий и процессоров число тестов, в которых каждый из рассмотренных подходов имеет преимущество, становится примерно равным. Можно рекомендовать решать задачу последовательно каждым из методов и выбирать лучшее решение.

С увеличением числа заданий метод ветвей и границ требует все больше времени на получение оптимального решения. Представляется оправданным и перспективным использовать данный метод для получения приближенного решения путем ограничения числа итераций на построение расписания при каждом заданном значении L. Применение метода ветвей и границ увеличило число построенных оптимальных расписаний для задачи $P|r_j|L_{\max}$ на 11% в среднем и составило 77.83% от общего количества тестов. Для 87% тестов приближенные решения, найденные с помощью метода ветвей и границ, имели относительную погрешность менее 5%.

Литература

1. *Brucker P.* Scheduling algorithms. Heidelberg: Springer, 2007. 378 p.

2. *Graham R. L., Lawner E. L., Kan R.* Optimization and approximation in deterministic sequencing and scheduling: A survey // Ann. of Disc. Math. 1979. Vol. 5, N 10. P. 287–326.

3. *Lenstra J. A., Kan R., Brucker P.* Complexity of machine scheduling problems // Ann. of Disc. Math. 1977. Vol. 1. P. 343–362.

4. *Brucker P, Garey M. R., Johnson D. S.* Scheduling equal-length tasks under tree-like precendence constraints to minimize maximum lateness // Matematics of Operations Research. 1977. N 2. P. 275–284.

5. *Zinder Y., Singh G.* Preemptive scheduling on parallel processors with due dates // Asia Pacific Journal of Operational Research. 2005. N 22. P. 445–462.

6. *Baker K. R.* Introduction to Sequencing and Scheduling. New York: John Wiley & Son, 1974. 318 p.

7. *Kanet J. J., Sridharan V.* Scheduling with inserted idle time: problem taxonomy and literature review // Operations Research. 2000. Vol. 48, N 1. P. 99–100.

8. *Carlier J.* The one-machine sequencing problem // European J. Oper. Res. 1982. N 11. P. 42–47.

9. *Григорьева Н. С.* Алгоритм ветвей и границ для задачи составления расписаний на параллельных процессорах // Вестн. С.-Петерб. ун-та. Сер. 10. Прикладная математика. Информатика. Процессы управления. 2009. Вып. 1. С. 44–55.

10. *Grigoreva N. S.* Multiprocessor scheduling with inserted idle time to minimize the maximum lateness // Proceedings of the 7th Multidisciplinary Intern. Conference of Scheduling: Theory and Applications. Prague: MISTA, 2015. P. 814–816.

11. *Gusfield D.* Bounds for naive multiple machine scheduling with release times and deadlines // Journal of Algorithms. 1984. Vol. 5. P. 1–6.

12. *Романовский И. В., Христова Н. П.* Решение минимаксных задач методом дихотомии // Журн. вычисл. математики и матем. физики. 1973. Т. 13, № 5. С. 200–209.

13. *Mastrolilli M.* Efficient approximation schemes for scheduling problems with release dates and delivery times // Journal of Scheduling. 2003. Vol. 6. P. 521–531.

14. *Fernandez E., Bussell B.* Bounds the number of processors and time for multiprocessor optimal schedules // IEEE Trans. on Computers. 1973. Vol. 4, N 11. P. 745–751.

Для цитирования: Григорьева Н. С. Задача минимизации максимального временного смещения для параллельных процессоров // Вестник Санкт-Петербургского университета. Сер. 10. Прикладная математика. Информатика. Процессы управления. 2016. Вып. 4. С. 51–65. DOI: 10.21638/11701/ spbu10.2016.405

References

1. Brucker P. *Scheduling algorithms.* Heidelberg, Springer Press, 2007, 378 p.

2. Graham R. L., Lawner E. L., Kan R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. of Disc. Math.*, 1979, vol. 5, no. 10, pp. 287–326.

3. Lenstra J. A., Kan R., Brucker P. Complexity of machine scheduling problems. *Ann. of Disc. Math.*, 1977, vol. 1, pp. 343–362.

4. Brucker P., Garey M. R., Johnson D. S. Scheduling equal-length tasks under tree-like precendence constraints to minimize maximum lateness. *Matematics of Operations Research*, 1977, no. 2, pp. 275–284.

5. Zinder Y., Singh G. Preemptive scheduling on parallel processors with due dates. *Asia Pacific Journal of Operational Research*, 2005, no. 22, pp. 445–462.

6. Baker K. R. *Introduction to Sequencing and Scheduling.* New York, John Wiley & Son Press, 1974, 318 p.

7. Kanet J. J., Sridharan V. Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, 2000, vol. 48, no. 1, pp. 99–100.

8. Carlier J. The one-machine sequencing problem. *European J. Oper. Res.*, 1982, no. 11, pp. 42–47.

9. Grigoreva N. S. Algoritm vetvey i granits dlya zadachi sostavleniya raspisaniya na parallelnykh protsessorakh [Branch and bound algorithm for multiprocessor scheduling problem]. *Vestnik of Saint Petersburg University. Series 10. Applied mathematics. Comruter science. Control processes*, 2009, issue 1, pp. 44–55. (In Russian)

10. Grigoreva N. S. Multiprocessor scheduling with inserted idle time to minimize the maximum lateness. *Proceedings of the 7th Multidisciplinary Intern. Conference of Scheduling: Theory and Applications.* Prague, MISTA Press, 2015, pp. 814–816.

11. Gusfield D. Bounds for naive multiple machine scheduling with release times and deadlines. *Journal of Algorithms*, 1984, vol. 5, pp. 1–6.

12. Romanovskii I. V., Khristova N. P. Reshenie minimaksnykh zadach metodom dikhotomii [The solution of minimax problem by the method of dichotomy]. *Computational mathematics and mathematical physics*, 1973, vol. 13, no. 5, pp. 200–209. (In Russian)

13. Mastrolilli M. Efficient approximation schemes for scheduling problems with release dates and delivery times. *Journal of Scheduling*, 2003, vol. 6, pp. 521–531.

14. Fernandez E., Bussell B. Bounds the number of processors and time for multiprocessor optimal schedules. *IEEE Trans. on Computers*, 1973, vol. 4, no. 11, pp. 745–751.

For citation: Grigoreva N. S. Scheduling problem to minimize the maximum lateness for parallel processors. *Vestnik of Saint Petersburg University. Series 10. Applied mathematics. Computer science. Control processes*, 2016, issue 4, pp. 51–65. DOI: 10.21638/11701/spbu10.2016.405

Статья рекомендована к печати проф. Л. А. Петросяном.

Статья поступила в редакцию 9 марта 2016 г.

Статья принята к печати 29 сентября 2016 г.